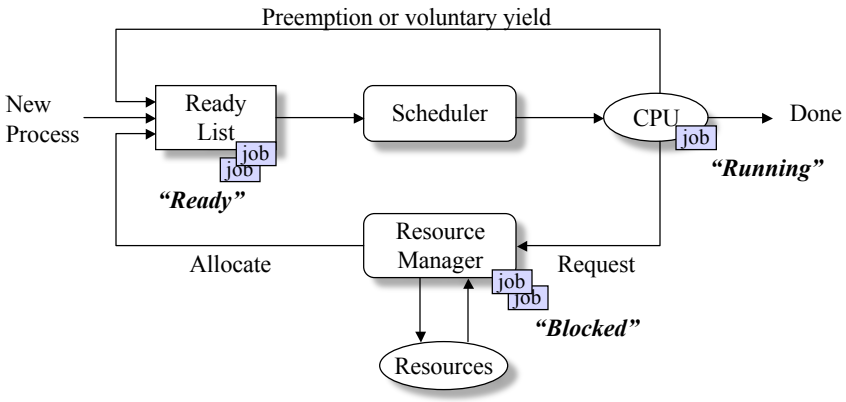
Slide 7-1

Scheduling: the act of sharing

Slide 7-2

Model of Process Execution



Thread leaves CPU

1. Completes execution
2. Requests a resource
3. Voluntarily release CPU
4. Involuntarily releases CPU (preempted)

Scheduling Policy - decides when thread leaves and which thread is selected to replace it

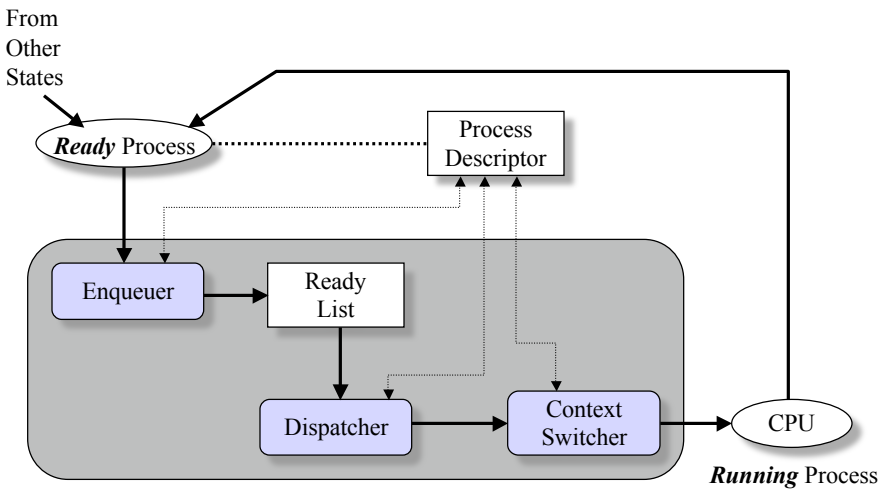
Scheduling mechanism - determines how the process manager carries out its duties

Scheduling Mechanism

- Depends on hardware
  - Need a clock device
  - Rest of scheduler implemented in OS software
- 3 logical parts
  1. Enqueuer
  2. Dispatcher
  3. Context switcher

The Scheduler

Slide 7-5



Scheduling Mechanism

Slide 7-6

1. Enqueuer

   Places a pointer to thread descriptor into ready list (priority placement may happen then or later when thread is selected)

2. Dispatcher

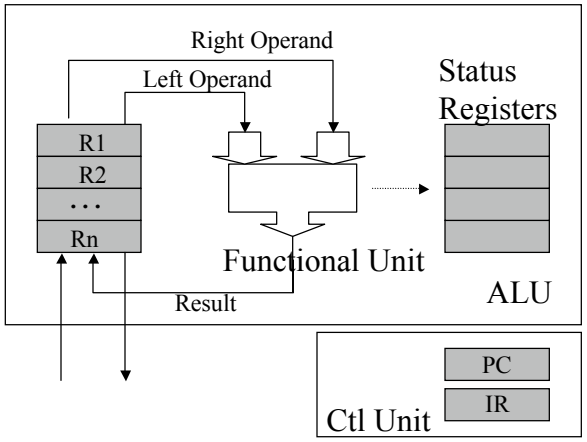
   Selects one ready thread and allocates CPU to that thread by performing context switch

3. Context switcher

   Saves the contents of all CPU registers (PC, IR, condition status) for the thread being moved out and places the new thread in.

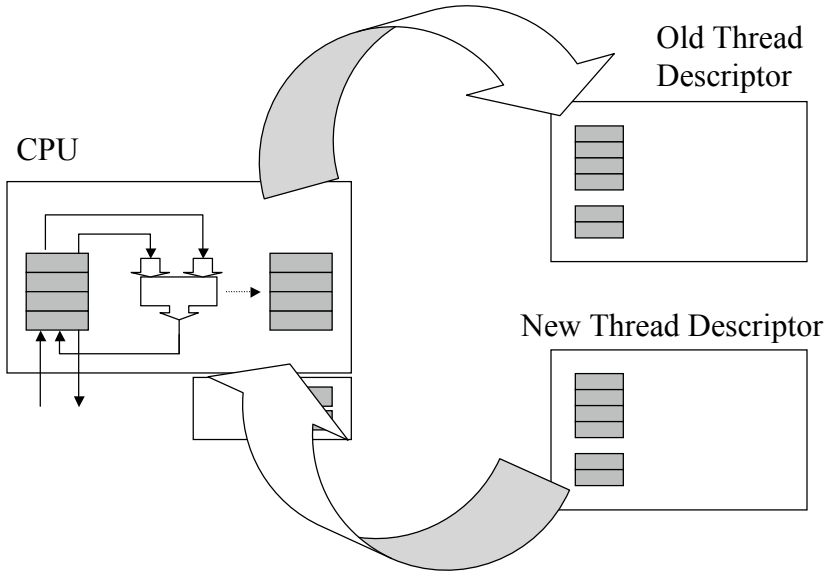
Process/Thread Context

Slide 7-7



Context Switching

Slide 7-8



Context Switching

- Switching Cost
  - Over 32 different register - 32 or 64 but registers
  - With conventional load and stores

(General registers + status registers)store ops X time/store

4 total context switches per multiplex

1. Save process 1's context
2. Load dispatcher's context
3. Save dispatcher's context
4. Load process 2's context

Context Switching

- Switching Cost
  - 50 nanoseconds to store 1 unit of data
  - 32 bit bus / 50 nanoseconds for each register
  - 32 general registers and 8 status registers
    - 40 x 50 nanaseconds = 2 microseconds
  - 2 microseconds more to load another thread
  - Total time: over 4 microseconds (including dispatch in and out)

1 GHz processor - 2 nanosecond instructions then CPU could execute about 2,000 instructions while switch is taking place.

Invoking the Scheduler

- Need a *mechanism* to call the scheduler
- Voluntary call
  - Process blocks itself
  - Calls the scheduler
- Involuntary call
  - External force (interrupt) blocks the process
  - Calls the scheduler

Voluntary CPU Sharing

Yield Instruction

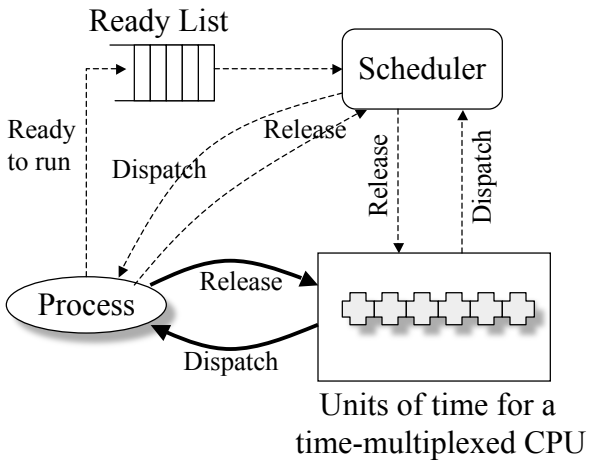
```
yield(pi.pc, pj.pc) {
    memory[pi.pc] = PC;
    PC = memory[pj.pc];
}
```

- p_i can be "automatically" determined from the processor status registers

```
yield(*, pj.pc) {
    memory[pi.pc] = PC;
    PC = memory[pj.pc];
}
```

Slide 7-13

Scheduler as CPU Resource Manager



Slide 7-14

More on Yield

- p_i and p_j can resume one another's execution

```
yield(*, pj.pc);
. . .
yield(*, pi.pc);
. . .
yield(*, pj.pc);
. . .
```

- Suppose p_j is the scheduler:

```
// p_i yields to scheduler
yield(*, pj.pc);
// scheduler chooses pk
yield(*, pk.pc);
// pk yields to scheduler
yield(*, pj.pc);
// scheduler chooses ...
```

Voluntary Sharing

Slide 7-15

Nonpreemptive Scheduler

- Scheduler using voluntary CPU sharing
- used in Xerox Alto PC
- used in earlier version of Mac OS

Problem:

What happens if some processes do not voluntarily yield?

Voluntary Sharing

Slide 7-16

- Every process must periodically yield to the scheduler
- Relies on correct process behavior
  - Malicious
  - Accidental (infinite loop)
- Need a mechanism to override running process

Involuntary CPU Sharing

- Interval timer
  - Device to produce a periodic interrupt
  - Programmable period

```
IntervalTimer() {
    InterruptCount--;
    if(InterruptCount <= 0) {
        InterruptRequest = TRUE;
        InterruptCount = K;
    }
}
                    SetInterval(programmableValue) {
                        K = programmableValue:
                        InterruptCount = K;
                        }
                    }
```

Involuntary CPU Sharing (cont)

- Interval timer device
  - Serviced by a timer handler
    - Keeps an in-memory clock up-to-date
    - Invokes the scheduler

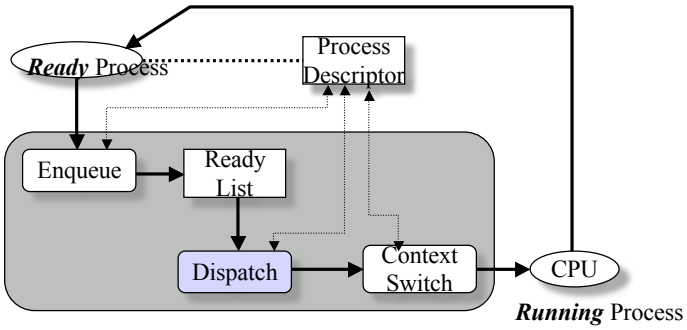
```
IntervalTimerHandler() {
    Time++;  // update the clock
    TimeToSchedule--; // update alarm
    if(TimeToSchedule <= 0) {
        <invoke scheduler>;
        TimeToSchedule = TimeSlice;
    }
}
```

Contemporary Scheduling

- Involuntary CPU sharing – timer interrupts
  - *Time quantum* determined by interval timer – usually fixed size for every process using the system
  - Sometimes called the *time slice length*

Preemptive Scheduler

Choosing a Process to Run



- *Mechanism* never changes
- Strategy = *policy* the dispatcher uses to select a process from the ready list
- Different policies for different requirements

Policy Considerations

- Policy can control/influence:
  - CPU utilization
  - Average time a process waits for service
  - Average amount of time to complete a job
- Could strive for any of:
  - Equitability (watch out for starvation)
  - Favor very short or long jobs
  - Meet priority requirements
  - Meet deadlines

Optimal Scheduling

- Suppose the scheduler knows each process p_i's *service time*, $\tau(p_i)$ -- or it can estimate each $\tau(p_i)$:
- Policy can optimize on any criteria, e.g.,
  - CPU utilization
  - Waiting time
  - Deadline (i.e. real-time system)
- To find an *optimal schedule*:
  - Must have a finite, fixed # of p_i
  - Know $\tau(p_i)$ for each p_i
  - Enumerate all schedules, then choose the best

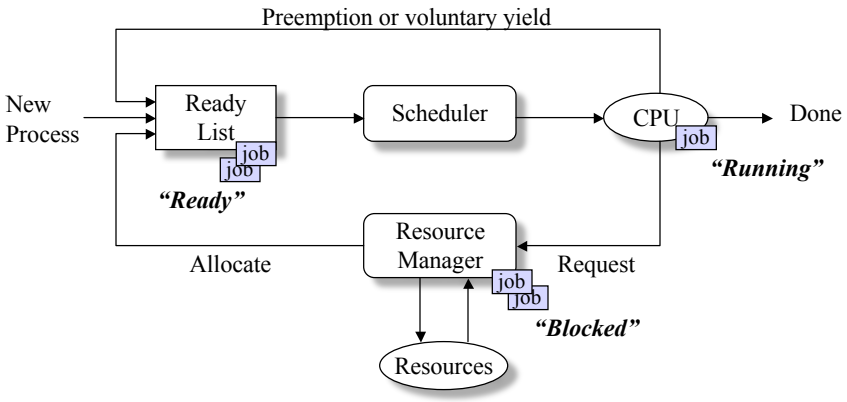
Slide 7-23

However ...

- The $\tau(p_i)$ are almost certainly just estimates
- General algorithm to choose optimal schedule is $O(n^2)$
- Other processes may arrive while these processes are being serviced
- Usually, optimal schedule is only a *theoretical benchmark* – scheduling policies try to *approximate* an optimal schedule

Slide 7-24

Model of Process Execution



Slide 7-25

Talking About Scheduling ...

- Let $P = \{p_i \mid 0 \leq i < n\}$ = set of processes
- Let $S(p_i) \in \{\text{running, ready, blocked}\}$

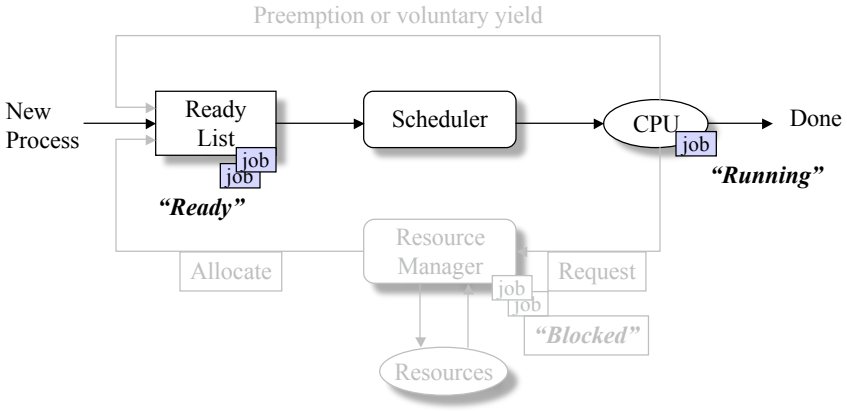
Common metrics for comparing scheduling strategies

1. Let $\tau(p_i)$ = Time process needs to be in running state (the *service time*)
2. Let $W(p_i)$ = Time p_i is in ready state before first transition to running (*wait time*)
3. Let $T_{TRnd}(p_i)$ = Time from p_i first enter ready to last exit running (*turnaround time*)

- Batch *Throughput rate* = inverse of avg T_{TRnd}
- Timesharing response time = $W(p_i)$

Slide 7-26

Simplified Model



- Simplified, but still can provide analysis result
- Easy to analyze performance
- No issue of voluntary/involuntary CPU sharing

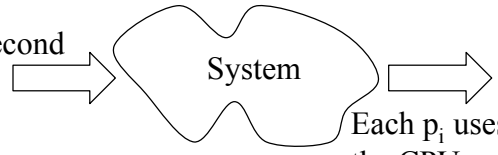
Slide 7-27

Estimating CPU Utilization

Let λ = the average rate at which processes are placed in the Ready List, *arrival rate*

Let μ = the average *service rate*
$\therefore$ $1/\mu$ = the average $\tau(p_i)$

λ p_i per second



Each p_i uses $1/\mu$ units of the CPU

Slide 7-28

Estimating CPU Utilization

Let λ = the average rate at which processes are placed in the Ready List, *arrival rate*

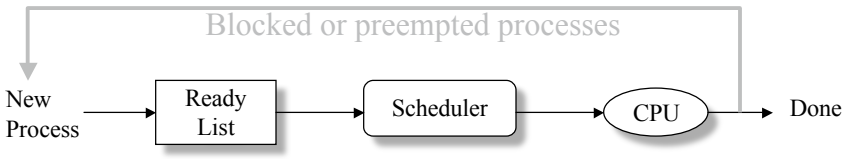
Let μ = the average *service rate*
$\therefore$ $1/\mu$ = the average $\tau(p_i)$

Let ρ = the fraction of the time that the CPU is expected to be busy
ρ = # p_i that arrive per unit time * avg time each spends on CPU
$\rho = \lambda * 1/\mu = \lambda/\mu$

- **Notice must have $\lambda < \mu$ (i.e., $\rho < 1$)**
- **What if ρ approaches 1?**

Nonpreemptive Schedulers

Slide 7-29

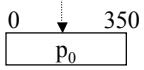


- Try to use the simplified scheduling model
- Only consider running and ready states
- Ignores time in blocked state:
  - "New process created when it enters ready state"
  - "Process is destroyed when it enters blocked state"
  - Really just looking at "small phases" of a process

Slide 7-30

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



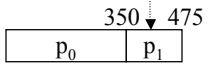
$T_{TRnd}(p_0) = \tau(p_0) = 350$

$W(p_0) = 0$

Slide 7-31

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



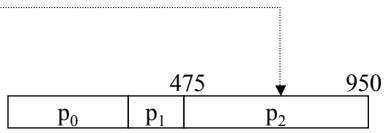
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$

Slide 7-32

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



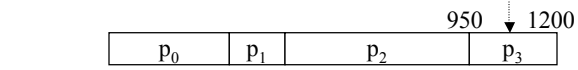
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$

Slide 7-33

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



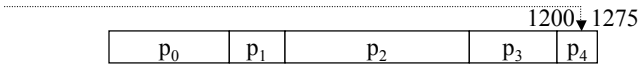
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$

Slide 7-34

First-Come-First-Served

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



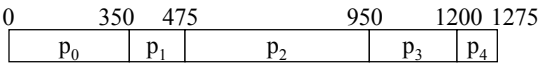
$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$
$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$
$W(p_4) = T_{TRnd}(p_3) = 1200$

FCFS Average Wait Time

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- *Easy to implement*
- Ignores service time, etc
- Not a great performer



$T_{TRnd}(p_0) = \tau(p_0) = 350$
$T_{TRnd}(p_1) = (\tau(p_1) + T_{TRnd}(p_0)) = 125+350 = 475$
$T_{TRnd}(p_2) = (\tau(p_2) + T_{TRnd}(p_1)) = 475+475 = 950$
$T_{TRnd}(p_3) = (\tau(p_3) + T_{TRnd}(p_2)) = 250+950 = 1200$
$T_{TRnd}(p_4) = (\tau(p_4) + T_{TRnd}(p_3)) = 75+1200 = 1275$

$W(p_0) = 0$
$W(p_1) = T_{TRnd}(p_0) = 350$
$W(p_2) = T_{TRnd}(p_1) = 475$
$W(p_3) = T_{TRnd}(p_2) = 950$
$W(p_4) = T_{TRnd}(p_3) = 1200$

$W_{avg} = (0+350+475+950+1200)/5 = 2974/5 = 595$
$T_{TRnd} = (350+475+950+1200+1275)/5 = 4250/5 = 850$

Predicting Wait Time in FCFS

- In FCFS, when a process arrives, all in ready list will be processed before this job
- Let μ be the service rate
- Let L be the ready list length
- $W_{avg}(p) = L*1/\mu + 0.5* 1/\mu = L/\mu+1/(2\mu)$

Slide 7-37

Predicting Wait Time in FCFS

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- Let's compare predicted wait with actual in earlier example:

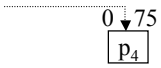
```
Average service time      = (350 + 125 + 475 + 250)/4
                          = 1200/4
                          = 300 time units

For the 4th process - L = 3 (3 in ready and 1 in CPU)
Estimated waiting time    = 3/(1/300) + 1/2(300)
                          = 900 + 150 = 1050
   (COMPARED TO ACTUAL WAIT TIME OF 1200)
```

Slide 7-38

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



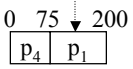
$T_{TRnd}(p_4) = \tau(p_4) = 75$

$W(p_4) = 0$

Slide 7-39

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



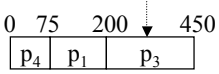
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Slide 7-40

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

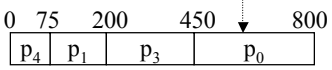
$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Slide 7-41

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $W(p_0) = 450$

$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

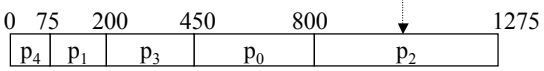
$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Slide 7-42

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$ $W(p_0) = 450$

$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$ $W(p_1) = 75$

$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$ $W(p_2) = 800$

$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$ $W(p_3) = 200$

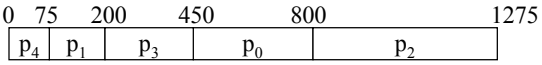
$T_{TRnd}(p_4) = \tau(p_4) = 75$ $W(p_4) = 0$

Slide 7-43

Shortest Job Next

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- *Minimizes wait time*
- May starve large jobs
- Must know service times



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 350+250+125+75 = 800$
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_4) = 125+75 = 200$
$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_0)+\tau(p_3)+\tau(p_1)+\tau(p_4) = 475+350+250+125+75 = 1275$
$T_{TRnd}(p_3) = \tau(p_3)+\tau(p_1)+\tau(p_4) = 250+125+75 = 450$
$T_{TRnd}(p_4) = \tau(p_4) = 75$

$W(p_0) = 450$
$W(p_1) = 75$
$W(p_2) = 800$
$W(p_3) = 200$
$W(p_4) = 0$

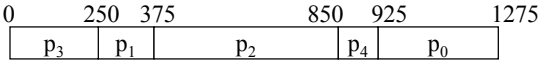
$W_{avg} = (450+75+800+200+0)/5 = 1525/5 = 305$

Slide 7-44

Priority Scheduling

i	$\tau(p_i)$	Pri
0	350	5
1	125	2
2	475	3
3	250	1
4	75	4

- *Reflects importance of external use*
- May cause starvation
- Can address starvation with aging



$T_{TRnd}(p_0) = \tau(p_0)+\tau(p_4)+\tau(p_2)+\tau(p_1))+\tau(p_3) = 350+75+475+125+250 = 1275$
$T_{TRnd}(p_1) = \tau(p_1)+\tau(p_3) = 125+250 = 375$
$T_{TRnd}(p_2) = \tau(p_2)+\tau(p_1)+\tau(p_3) = 475+125+250 = 850$
$T_{TRnd}(p_3) = \tau(p_3) = 250$
$T_{TRnd}(p_4) = \tau(p_4)+ \tau(p_2)+ \tau(p_1)+\tau(p_3) = 75+475+125+250 = 925$

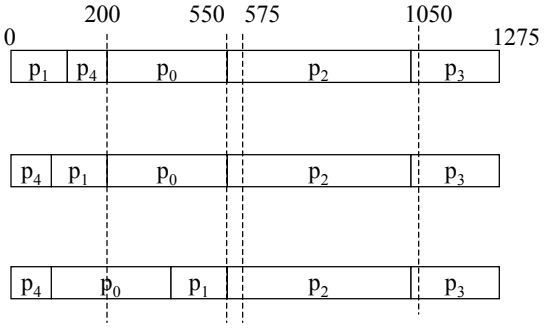
$W(p_0) = 925$
$W(p_1) = 250$
$W(p_2) = 375$
$W(p_3) = 0$
$W(p_4) = 850$

$W_{avg} = (925+250+375+0+850)/5 = 2400/5 = 480$

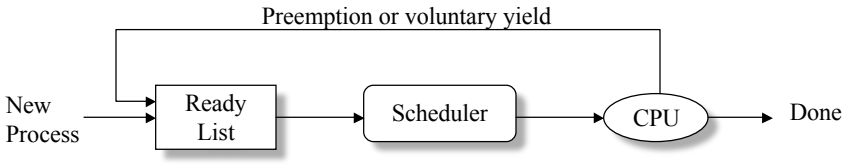
Deadline Scheduling

i	$\tau(p_i)$	Deadline
0	350	575
1	125	550
2	475	1050
3	250	(none)
4	75	200

- *Allocates service by deadline*
- May not be feasible



Preemptive Schedulers



- Highest priority process is guaranteed to be running at all times
  - Or at least at the beginning of a time slice
- Dominant form of contemporary scheduling
- But complex to build & analyze

Slide 7-47

Round Robin (TQ=50)

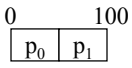
i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

$W(p_0) = 0$

Slide 7-48

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

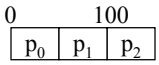


$W(p_0) = 0$
$W(p_1) = 50$

Slide 7-49

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

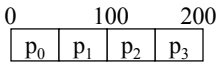


$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$

Slide 7-50

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

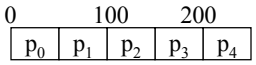


$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$

Slide 7-51

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

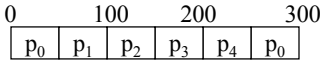


$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Slide 7-52

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

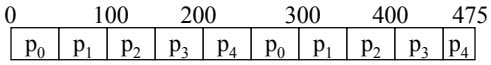


$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Slide 7-53

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



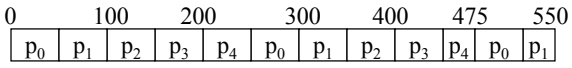
$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

$T_{TRnd}(p_4) = 475$

Slide 7-54

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

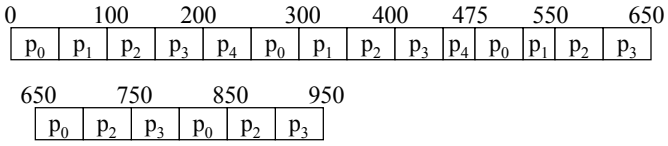
$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_4) = 475$

Slide 7-55

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_3) = 950$

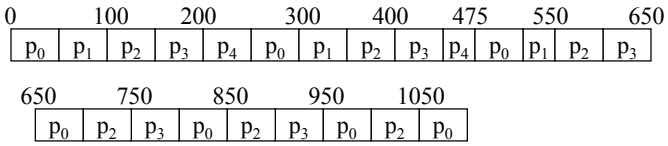
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Slide 7-56

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = 1100$

$T_{TRnd}(p_1) = 550$

$T_{TRnd}(p_3) = 950$

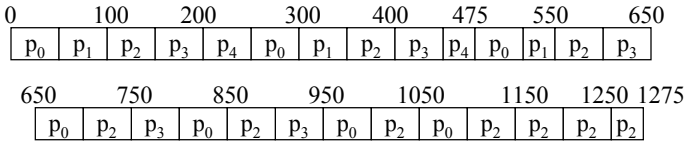
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Slide 7-57

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75



$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$
$T_{TRnd}(p_2) = 1275$
$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

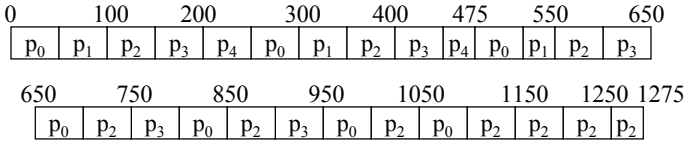
$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

Slide 7-58

Round Robin (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

- Equitable
- Most widely-used
- Fits naturally with interval timer



$T_{TRnd}(p_0) = 1100$
$T_{TRnd}(p_1) = 550$
$T_{TRnd}(p_2) = 1275$
$T_{TRnd}(p_3) = 950$
$T_{TRnd}(p_4) = 475$

$W(p_0) = 0$
$W(p_1) = 50$
$W(p_2) = 100$
$W(p_3) = 150$
$W(p_4) = 200$

$T_{TRnd\_avg} = (1100+550+1275+950+475)/5 = 4350/5 = 870$

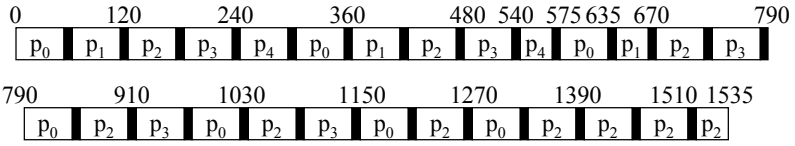
$W_{avg} = (0+50+100+150+200)/5 = 500/5 = 100$

Slide 7-59

RR with Overhead=10 (TQ=50)

i	$\tau(p_i)$
0	350
1	125
2	475
3	250
4	75

•Overhead must be considered



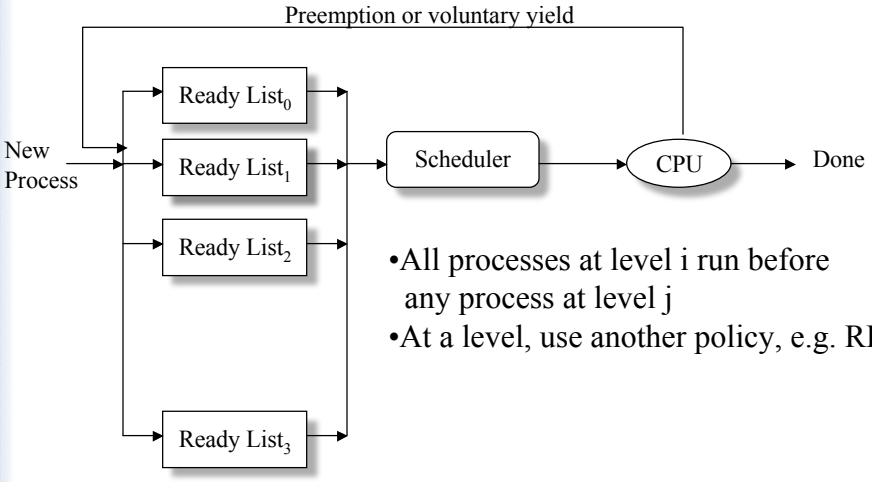
$T_{TRnd}(p_0) = 1320$ $W(p_0) = 0$
$T_{TRnd}(p_1) = 660$ $W(p_1) = 60$
$T_{TRnd}(p_2) = 1535$ $W(p_2) = 120$
$T_{TRnd}(p_3) = 1140$ $W(p_3) = 180$
$T_{TRnd}(p_4) = 565$ $W(p_4) = 240$

$T_{TRnd\_avg} = (1320+660+1535+1140+565)/5 = 5220/5 = 1044$

$W_{avg} = (0+60+120+180+240)/5 = 600/5 = 120$

Multi-Level Queues

Slide 7-60



•All processes at level i run before any process at level j
•At a level, use another policy, e.g. RR

Slide 7-61

Contemporary Scheduling

- Involuntary CPU sharing -- timer interrupts
  - *Time quantum* determined by interval timer -- usually fixed for every process using the system
  - Sometimes called the *time slice length*
- *Priority*-based process (job) selection
  - Select the highest priority process
  - Priority reflects policy
- With *preemption*
- Usually a variant of *Multi-Level Queues*

Slide 7-62

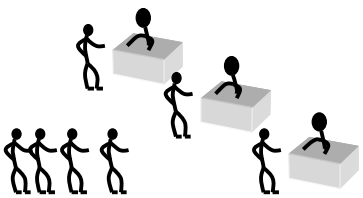
BSD 4.4 Scheduling

- Involuntary CPU Sharing
- Preemptive algorithms
- 32 Multi-Level Queues
  - Queues 0-7 are reserved for system functions
  - Queues 8-31 are for user space functions
  - `nice` influences (but does not dictate) queue level

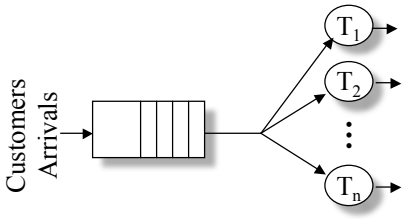
Windows NT/2K Scheduling

- Involuntary CPU Sharing across threads
- Preemptive algorithms
- 32 Multi-Level Queues
  - Highest 16 levels are "real-time"
  - Next lower 15 are for system/user threads
    - Range determined by process base priority
  - Lowest level is for the idle thread

Bank Teller Simulation



Tellers at the Bank



Model of Tellers at the Bank

Simulation Kernel Loop

Slide 7-65

```
simulated_time = 0;
while (true) {
      event = select_next_event();
      if (event->time > simulated_time)
            simulated_time = event->time;
      evaluate(event->function, …);
}
```

Simulation Kernel Loop(2)

Slide 7-66

```
void runKernel(int quitTime)
{
        Event *thisEvent;

// Stop by running to elapsed time, or by causing quit execute
        if(quitTime <= 0) quitTime = 9999999;
        simTime = 0;
        while(simTime < quitTime) {
        // Get the next event
                if(eventList == NIL) {
                // No more events to process
                        break;
                }
                thisEvent = eventList;
                eventList = thisEvent->next;
                simTime = thisEvent->getTime(); // Set the time
        // Execute this event
                thisEvent->fire();
                delete(thisEvent);
        };
}
```